UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO DE FIN DE GRADO

DISEÑO Y DESARROLLO DE UNA HERRAMIENTA PARA ANÁLISIS FORENSE DE BACKUPS DE DISPOSITIVOS IPHONE

Grado en Ingeniería Informática

Adrián Señor Palazuelo

Tutor: Álvaro Ortigosa Juárez

Julio 2014

Resumen

Resumen

El uso de los dispositivos móviles se extiende día a día convirtiéndose en el medio principal de la mayoría de las personas para comunicarse, gestionar sus tareas diarias, guardar información personal o gastar su tiempo de ocio. Es increible la cantidad de información que puede acumular a lo largo de unas pocas semanas. Esto hace que se valore tanto el poder recuperarla o analizarla. El iPhone es uno de los dispositivos móviles mas vendidos y existe una aplicación, iTunes, que es capaz de crear copias de seguridad de éste. En este trabajo se pretende dar una solución a la lectura y análisis de este tipo de copias de seguridad desarrollando una aplicación que sirva como base para futuros desarrollos, que permita a cualquiera ampliar la funcionalidad de ésta fácilmente sin tener que conocer todo el funcionamiento al detalle. Además, para poder examinar copias de seguridad cifradas, se desarrollan mecanismos de ataque para averiguar la contraseña utilizada.

Palabras clave

Copia de seguridad, iPhone, iTunes, Python, wxPython, componente

Abstract

The use of mobile devices is spreading day by day becoming the primary device of most people to communicate, manage their everyday tasks, save personal information or spend their leisure time. It's amazing the amount of information that can accumulate over a few weeks. This makes very valuable to retrieve or analyze it. The iPhone is one of the best selling mobile devices and there is an application, iTunes, that is able to create backup copies of it. This work aims to provide a solution to read and analyze this such of backups developing an application that serves as a basis for future developments, which allows anyone to extend the functionality easily without having to know all the workings in detail. In addition, to examine encrypted backups, attack mechanisms are developed to find out the password used.

Key words

Backup, iPhone, iTunes, Python, wxPython, plugin

Índice general

Índice de figuras

Índice de tablas

1

Introducción

Cada día más personas disponen de dispositivos móviles que utilizan durante todo el día. Los utilizan para comunicarse con su pareja, para apuntar los productos que tendrán que comprar en el supermercado o para recordar las reuniones que tendrán durante el día. Pero también se utilizan para guardar las contraseñas de los diferentes servicios que utilizan, información bancaria o los datos de sus tarjetas de débito. Toda esta información queda almacenada en el dispositivo y con las herramientas adecuadas se puede recuperar.

Muchas personas pierden sus dispositivos móviles o sufren un robo y tienen información muy importante que necesitan recuperar. Hay profesionales forenses que se dedican a analizar esta información en busca de una evidencia que haga a una persona culpable o inocente. Siempre hay alguien interesado en recuperar la información personal almacenada en un dispositivo móvil.

Debido a que la cantidad de información es enorme y los modos de almacenarla son muy variados, en este proyecto se trata de desarrollar una arquitectura que facilite el análisis de la información contenida en una copia de seguridad de un dispositivo iPhone.

1.1 Motivación

El iPhone es uno de los dispositivos móviles más vendidos del mundo y de los que mas tráfico de internet acapara. No es extraño que al llevarlo siempre consigo sea el principal modo de comunicación y almacenamiento de información de la mayoría de las personas. Además por miedo a perder los datos almacenados suelen realizar copias de seguridad. Aquí reside la motivación de este proyecto, poder recuperar o analizar toda esa información guardada.

1.2 Objetivos

El objetivo de este proyecto es desarrollar una arquitectura base para una aplicación con interfaz gráfica que permita analizar copias de seguridad de dispositivos iPhone. Ya que puede contener una gran cantidad de información muy heterogénea, tiene que ser fácilmente ampliable o modificable. Puede ser utilizable tanto por usuarios que solo pretendan recuperar información como por usuarios avanzados que quieran analizar información en profundidad.

Esta aplicación tiene que realizar las siguientes tareas:

- Localizar copias de seguridad de dispositivos iPhone almacenadas en un equipo
- Descifrar aquellas copias de seguridad que se encuentren cifradas
- Extraer información de los archivos encontrados

1.3 Estructura del documento

Este documento se divide en 6 capítulos, de los cuales este es el primero. El segundo capítulo sitúa al lector en el contexto del proyecto y analiza las herramientas existentes. En el tercer capítulo se describe el análisis y el diseño propuesto para la aplicación. El cuarto capítulo trata de las tecnologías empleadas, como el lenguaje de programación y la interfaz gráfica utilizados. En el quinto capítulo se muestra la implementación que se ha llevado a cabo con detalles y muestras de ejemplo. Por último en el sexto capítulo se exponen las conclusiones finales de este proyecto y se enumeran una serie de desarrollos que se podrían realizar en un futuro.

2

Estado del arte

2.1 Punto de partida

Todo comienza por un usuario que conecta su dispositivo iPhone a su ordenador, abre iTunes y realiza una copia de seguridad de su dispositivo. Este usuario puede elegir si cifrar la copia de seguridad o no, en caso afirmativo debe introducir la contraseña que desee.

iTunes[1] es la aplicación utilizada para crear copias de seguridad de dispositivos iOS, está desarrollada por Apple, es de código propietario y actualmente soporta OS X y Windows.

Guarda todas las copias de seguridad que crea en una carpeta específica del sistema. La localización de esta carpeta es diferente según el sistema operativo:

- **Windows XP:** C:Users\<nombre_de_usuario>\Application Data\Apple Computer\MobileSync\Backup\<UDID>
- **Windows Vista/7/8:** C:Users\<nombre_de_usuario>\AppData\Roaming\Apple Computer\MobileSync\Backup\<UDID>
- **OS X:** ~/Library/Application Support/MobileSync/Backup/<UDID>

Dentro de esta carpeta se crea una carpeta con el UDID del dispositivo como nombre. De esta manera se identifica unívocamente la copia de seguridad de cada dispositivo. UDID es la abreviatura de Unique Device Identifier, es un identificador único que se asigna a cada dispositivo iOS. Tiene un tamaño de 20 bytes. Un ejemplo de UDID sería b9242828d4572e6d4a5a7942131efe66b1b11d85. La estructura de esta carpeta se describirá en el apartado 2.3 Estructura de una copia de seguridad.

[1] https://www.apple.com/itunes/

2.2 Herramientas existentes

Se han seleccionado las principales herramientas existentes para el análisis de copias de seguridad de dispositivos iPhone. A continuación se muestra un resumen de sus características:

	iPhone Backup Extractor[2]	**iPhone Backup Extractor**[3]	**iBackup Extractor**[4]	**iPhone Backup Analyzer**[5]	**Elcomsoft iOS Forensic Toolkit**[6]
Gratuita	✗	✓	✗	✓	✗
Código libre	✗	✗	✗	✓	✗
Extensible	✗	✗	✗	✓	✗
Descifrado	✓	✓	✓	✗	✓
Ataques para averiguar la contraseña	✗	✗	✗	✗	✓
Recuperación de archivos	✓	✓	✓	✗	✓
Previsualización de archivos	✗	✓	✓	✓	✗
Análisis avanzado de información del usuario	✗	✗	✗	✓	✗
Interfaz gráfica	✓	✓	✓	✓	✗

Tabla 1. Características de las herramientas existentes

Como se puede observar, ninguna de las herramientas analizadas cumple todas las características deseadas. Las tres primeras herramientas están enfocadas a usuarios que quieren recuperar o extraer información básica y la última está enfocada a usuarios avanzados que quieren obtener la contraseña de una copia de seguridad cifrada. La que más se acerca a nuestro objetivo es la cuarta, iPhone Backup Analyzer, pero carece de descifrado de copias de seguridad y la implementación de ataques, bien sea por diccionario o por fuerza bruta, para averiguar la contraseña.

[2] http://www.iphonebackupextractor.com
[3] http://www.iphonebackup-extractor.com
[4] http://www.wideanglesoftware.com/ibackupextractor/
[5] http://www.ipbackupanalyzer.com
[6] http://www.elcomsoft.com/eift.html

2.3 Estructura de una copia de seguridad

Dentro de la carpeta de cada copia de seguridad que crea iTunes se encuentran cuatro archivos donde se almacenan datos relativos al propio dispositivo y a la copia de seguridad: Info.plist, Status.plist, Manifest.plist y Manifest.mbdb. El resto de archivos que se pueden encontrar son los propios archivos que han sido salvaguardados del dispositivo.

Los archivos .plist representan archivos serializados cuya estructura suele ser pares "clave, valor" que relacionan una propiedad y su valor. También permiten agrupar pares "clave, valor" en listas.

2.3.1 Info.plist

Contiene información relacionada con el dispositivo como la versión del sistema operativo, IMEI, número de teléfono, modelo, número de serie, etc.

2.3.2 Status.plist

Contiene información sobre el estado de la copia de seguridad, la fecha de realización, si se trata de una copia de seguridad completa, el UUID (Universally Unique IDentifier) y la versión.

2.3.3 Manifest.plist

Contiene el conjunto de claves que se utilizan para descifrar los archivos, si el dispositivo tiene una clave de acceso, si la copia de seguridad se encuentra cifrada, el listado de aplicaciones instaladas en el dispositivo e información sobre la copia de seguridad y el dispositivo.

2.3.4 Manifest.mbdb

Fichero binario que contiene la información de los ficheros que contiene la copia de seguridad en una secuencia de registros.

La cabecera:

```
mbdb\x05\x00
```

Le sigue la información de cada registro, de tamaño variable, que está formado por:

Tipo	Descripción
string	Dominio de la aplicación
string	Ruta del fichero en el dispositivo
string	Destino si se trata de un enlace simbólico
string	Hash (SHA-1) de los contenidos del fichero
string	Clave de cifrado (0xFFFF si no está cifrado)
uint32	Permisos del fichero
uint32	Desconocido
uint32	Desconocido
uint32	Nodo índice
uint32	Id del usuario propietario
uint32	Id del grupo propietario
uint32	Fecha de la última modificación
uint32	Fecha del último acceso
uint32	Fecha de la última modificación del estado
uint64	Tamaño del fichero (0 si es un directorio o un enlace simbólico)
uint8	Clave de protección
uint8	Número de propiedades que siguen a continuación

Tabla 2. Estructura de un registro de un fichero mbdb

Cada propiedad está definida como un par de cadenas de texto:

Tipo	Descripción
string	nombre de la propiedad
string	puede ser una cadena de texto o un contenido binario

Tabla 3. Estructura de una propiedad de un registro de un fichero mbdb

Todos los valores están en formato big endian, las cadenas de texto empiezan con un entero sin signo de 2 bytes que contiene la longitud en caracteres y le sigue el contenido o empiezan con 0xFFFF si no hay información.

2.3.5 Resto de archivos

El resto de archivos que se encuentran son los propios del dispositivo, se encuentran todos al mismo nivel, no existe ninguna jerarquía de carpetas y están nombrados con un hash que se calcula de la siguiente forma:

```
SHA-1(<Domino>-<Ruta>)
```

La única forma de saber qué archivo es exactamente es leer el registro de Manifest.mbdb y obtener de él toda la información.

A continuación se muestran algunos ejemplos:

· **Base de datos que almacena los contactos**

Hash: 31bb7ba8914766d4ba40d6dfb6113c8b614be442

Dominio: HomeDomain

Ruta: Library/AddressBook/AddressBook.sqlitedb

· **Base de datos que almacena las cuentas personales añadidas**

Hash: 943624fd13e27b800cc6d9ce1100c22356ee365c

Dominio: HomeDomain

Ruta: Library/Accounts/Accounts3.sqlite

· **Base de datos que almacena el historial de mensajes de Whatsapp**

Hash: 1b6b187a1b60b9ae8b720c79e2c67f472bab09c0

Dominio: AppDomain-net.whatsapp.WhatsApp

Ruta: Documents/ChatStorage.sqlite

· **Fotografía almacenada en el carrete**

Hash: eabc87a1d86f0a4b083b6088ff307ed035611af4

Dominio: CameraRollDomain

Ruta: Media/DCIM/100APPLE/IMG_0051.JPG

2.3.6 Cifrado

Si la copia de seguridad se encuentra cifrada, los archivos del dispositivo se encontrarán cifrados, mientras que los archivos relacionados con la información de la copia de seguridad no lo estarán. Así se puede seguir extrayendo toda la información principal contenida en Info.plist, Status.plist y Manifest.plist.

Para leer cualquier otro fichero necesitaremos Manifest.plist y Manifest.mbdb. Del primero obtendremos un conjunto de claves y del segundo obtendremos qué clave será la necesaria para descifrar el archivo.

En la lista de propiedades Manifest.plist encontramos un conjunto de claves cuyo identificador es "BackupKeyBag" y su valor es un objeto binario que contiene una serie de registros con la siguiente estructura:

Tipo	Descripción
string	identificador
uint32	tamaño de los datos en bytes
bytes	datos

Tabla 4. Estructura de un registro del conjunto de claves

Siempre empieza con un registro identificado como "VERS" que especifica la versión utilizada representar este conjunto de claves. Actualmente la versión utilizada es la 3.

En el caso de las copias de seguridad cifradas, las claves que se utilizan para descifrar los archivos estarán cifradas en el conjunto de claves. Para cifrarlas utiliza PBKDF2[7] (Password-Based Key Derivation Function 2) con mil iteraciones. PBKDF2 es una función derivadora de claves, a partir de una clave y una sal[8] aplica un algoritmo de hashing un número de iteraciones especificado. En este caso la clave es la que introdujo el usuario en iTunes, la sal se obtiene del conjunto de claves y el algoritmo de hashing utilizado es SHA-1. De esta manera averiguar la contraseña que utilizó el usuario se hace mucho mas compleja, el coste computacional de mil iteraciones asciende a unas treinta veces el coste de una iteración.

Una vez que se descifran estas claves, a partir del archivo Manifest.mbdb se obtiene la clave necesaria para descifrar el archivo deseado. Los archivos se encuentran cifrados usando el algoritmo AES[9] (Advanced Encryption Standard) en modo CBC[10] (Cipher Block Chaining).

[7] http://en.wikipedia.org/wiki/PBKDF2
[8] http://es.wikipedia.org/wiki/Sal_(criptograf%C3%ADa)
[9] http://es.wikipedia.org/wiki/Advanced_Encryption_Standard
[10] http://es.wikipedia.org/wiki/Cifrado_por_bloques

3

Análisis y diseño

3.1 Requisitos

3.1.1 Requisitos funcionales

RF1 Localización de copias de seguridad

La aplicación debe ser capaz de buscar copias de seguridad en las carpetas predeterminadas de iTunes. De igual manera debe permitir al usuario seleccionar una localización.

RF2 Descifrado de copias de seguridad

Debe poder descifrar copias de seguridad.

RF3 Ataques para averiguar la contraseña de una copia de seguridad

Debe poder realizar ataques que permitan averiguar cuál es la contraseña que se ha utilizado para cifrar una copia de seguridad

RF4 Análisis de ficheros

Debe ser capaz de identificar los diferentes ficheros contenidos en una copia de seguridad.

3.1.1 Requisitos no funcionales

RNF1 Soporte para Windows y OS X

Debe soportar ambas plataformas, en las que se encuentra presente iTunes.

RNF2 Sencilla y extensible

La aplicación debe permanecer lo mas sencilla posible para garantizar que cualquier persona pueda ampliar sus capacidades.

RNF3 Interfaz gráfica

La aplicación debe tener una interfaz gráfica de usuario con una buena usabilidad.

RNF4 No alteración de los datos originales

Cualquier tipo de operación que se realice no debe modificar los archivos originales.

RNF5 Para todo tipo de usuarios

Debe estar orientada tanto a usuarios finales como usuarios avanzados.

3.2 Arquitectura

Una vez definidos los requisitos de la aplicación se procede al análisis y diseño de la arquitectura que se va a desarrollar.

Se identifican tres objetos iniciales para poder leer una copia de seguridad: la propia copia de seguridad, el fichero Manifest.mbdb y un registro del fichero anterior. Para mantener una arquitectura extensible lo mejor es el desarrollo utilizando plugins, complementos que añaden nueva funcionalidad.

Esta es la arquitectura propuesta:

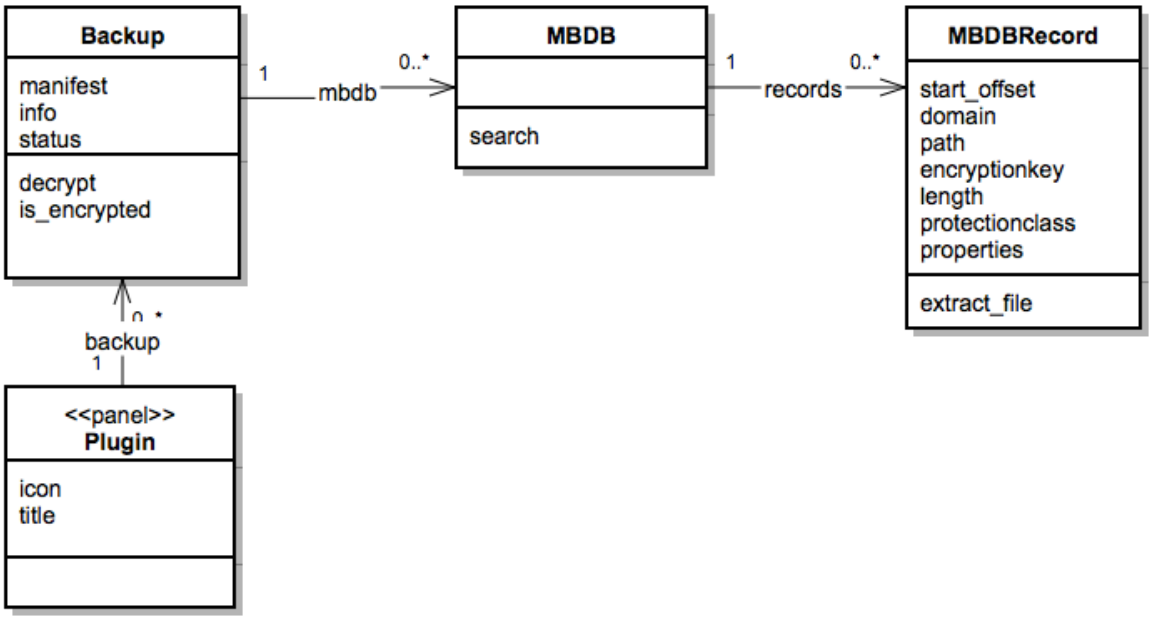

Ilustración 1. Arquitectura

Con esta arquitectura tenemos lo estrictamente necesario para poder abstraer una copia de seguridad e identificar los ficheros que contiene. Lo que se ha considerado más

importante es mantener la sencillez y poder incrementar la complejidad por medio de plugins.

La clase Backup tiene referencias a los cuatro archivos importantes: Info.plist, Status.plist, Manifest.plist y Manifest.mbdb. Permite dos operaciones: comprobar si está cifrada y descifrar.

La clase MBDB abstrae el fichero Manifest.mbdb, especificando la ruta a este fichero lee los datos de éste y crea los registros. Tener esta información en memoria permite realizar búsquedas de ficheros de forma más rápida con la operación de búsqueda.

La clase MBDBRecord representa un registro del fichero Manifest.mbdb y guarda la información relevante de él. Solo permite la extracción del fichero a la ubicación seleccionada.

Por último tenemos la clase Plugin, como el objetivo de esta aplicación es contar con una interfaz gráfica lo mejor es que herede de una ventana o panel, componentes visuales básicos de cualquier interfaz gráfica actual. Para diferenciarlos cada uno cuenta con un título y un icono. Cada plugin necesita obtener el Backup para buscar los ficheros que necesite y luego mostrar información.

Además, se desarrollará un plugin como ejemplo que permita la visualización del contenido de una base de datos. En un dispositivo como el iPhone el sistema de gestión de bases de datos más utilizado es SQLite[11]. Para hacer más sencilla su manipulación y añadir funcionalidad extra se propone la siguiente clase:

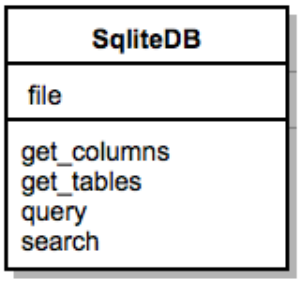

Ilustración 2. SQLiteDB

Esta clase se ha diseñado como una fachada de una base de datos SQLite. Las consultas para obtener la información de las tablas y las columnas que tienen son muy específicas y se pueden simplificar implementando dos operaciones que devuelvan esta información. A su vez tiene que mantener la operación de ejecutar una consulta. Para realizar un completo análisis de la información que contiene se ha añadido una operación para buscar un texto especificado, busca primero qué columnas contienen

[11] http://www.sqlite.org

texto, luego prepara una consulta en la que se busca el texto en cada columna y por último devuelve los resultados obtenidos.

3.2 Diseño

Se distinguen tres pantallas diferentes para la aplicación, cada una atiende a un objetivo o requisito, que son: una pantalla inicial, otra pantalla para el descifrado y la vista final con los plugins.

La pantalla inicial tendrá el siguiente aspecto:

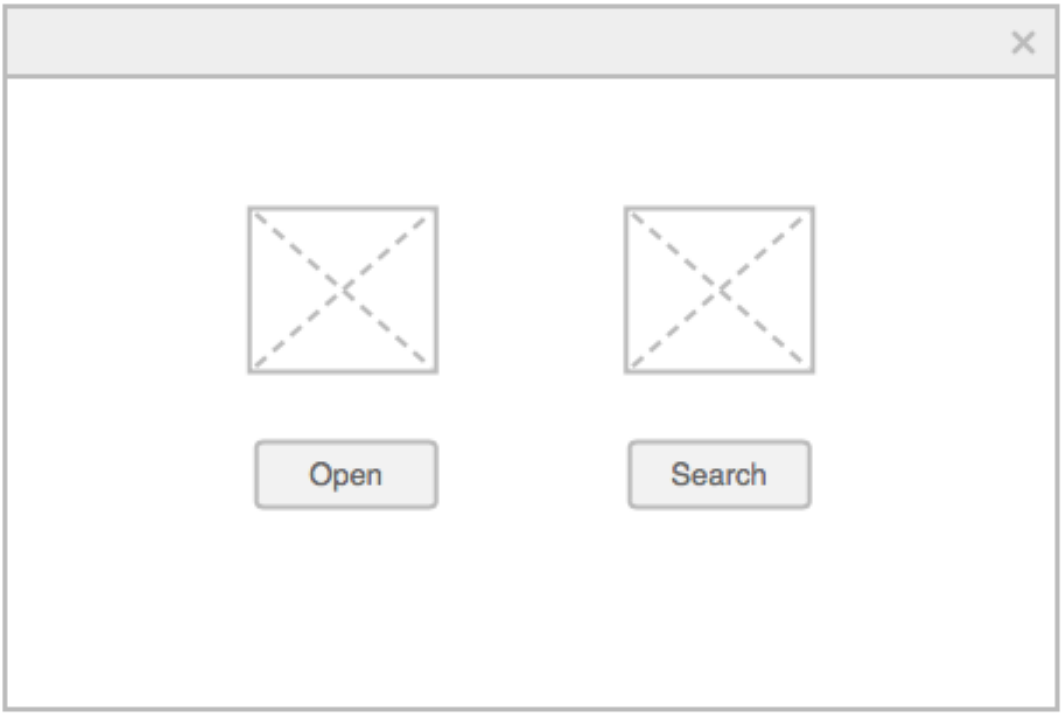

Ilustración 3. Prototipo de la pantalla inicial

Desde el primer botón el usuario puede navegar por el sistema de ficheros y seleccionar una copia de seguridad. Con el segundo botón la aplicación busca si existen copias de seguridad en las carpetas predefinidas del sistema.

Si la copia de seguridad está cifrada pasaremos a la pantalla de descifrado:

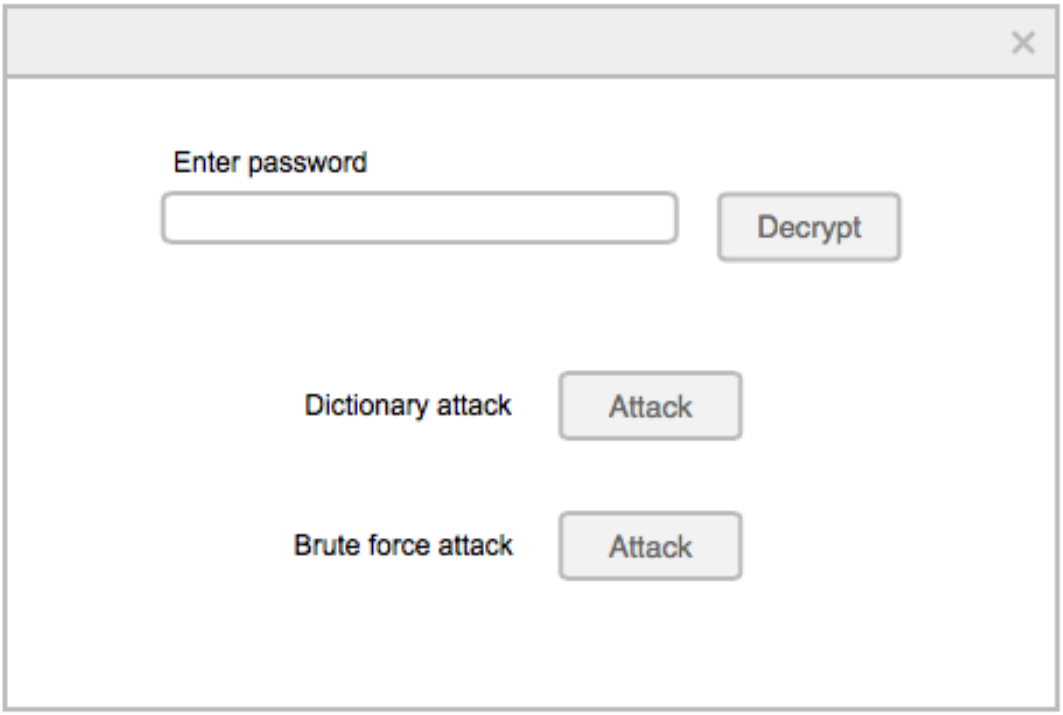

Ilustración 4. Prototipo de la pantalla de descifrado

Tendríamos tres opciones, en la primera se puede introducir la contraseña directamente, la segunda realiza un ataque por diccionario y en la tercera se ejecuta un ataque por fuerza bruta.

Después de abrir una copia de seguridad se mostrará la vista principal donde se podrán ver los plugins disponibles. En la herramienta analizada iPhone Backup Analyzer se ha utilizado un diseño en el que los plugins se muestran en la barra de menús y cada uno se abre en una ventana independiente, es muy útil si queremos ver la información de varios plugins a la vez, pero también se puede volver caótico si abrimos demasiados y luego queremos buscar uno en concreto. En vez de utilizar el método anterior se propone un diseño en el que se puedan ver de una manera gráfica los plugins existentes y solo se pueda seleccionar uno a la vez para su visualización. Se crearán todos los plugins al inicio, de esta manera aunque nos movamos de uno a otro siempre estarán ejecutando todos en segundo plano, es muy útil si ejecutamos un proceso largo en un plugin y mientras queremos consultar información de otro.

El diseño propuesto es el siguiente:

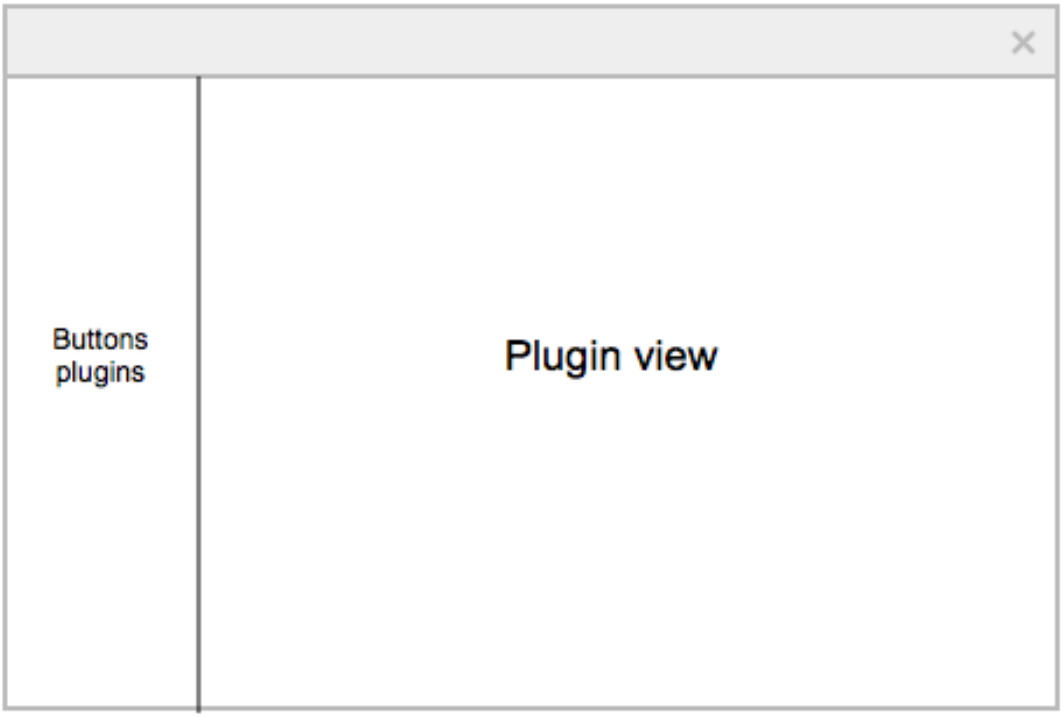

Ilustración 5. Prototipo de la estructura de la pantalla principal

Tiene dos zonas, un lateral con los botones de los plugins y una zona principal donde se muestra el plugin activo.

En el lateral se mostrará un botón por cada plugin disponible, si el listado excede el tamaño visible se podrá desplazar verticalmente mostrando el resto de ellos. Cada botón se mostrará con el icono especificado en el plugin.

En la zona principal siempre se mostrará el plugin seleccionado.

A continuación se muestra un ejemplo con cuatro plugins y un plugin activo que muestra el nombre del dispositivo, el modelo y la fecha de realización de la copia de seguridad:

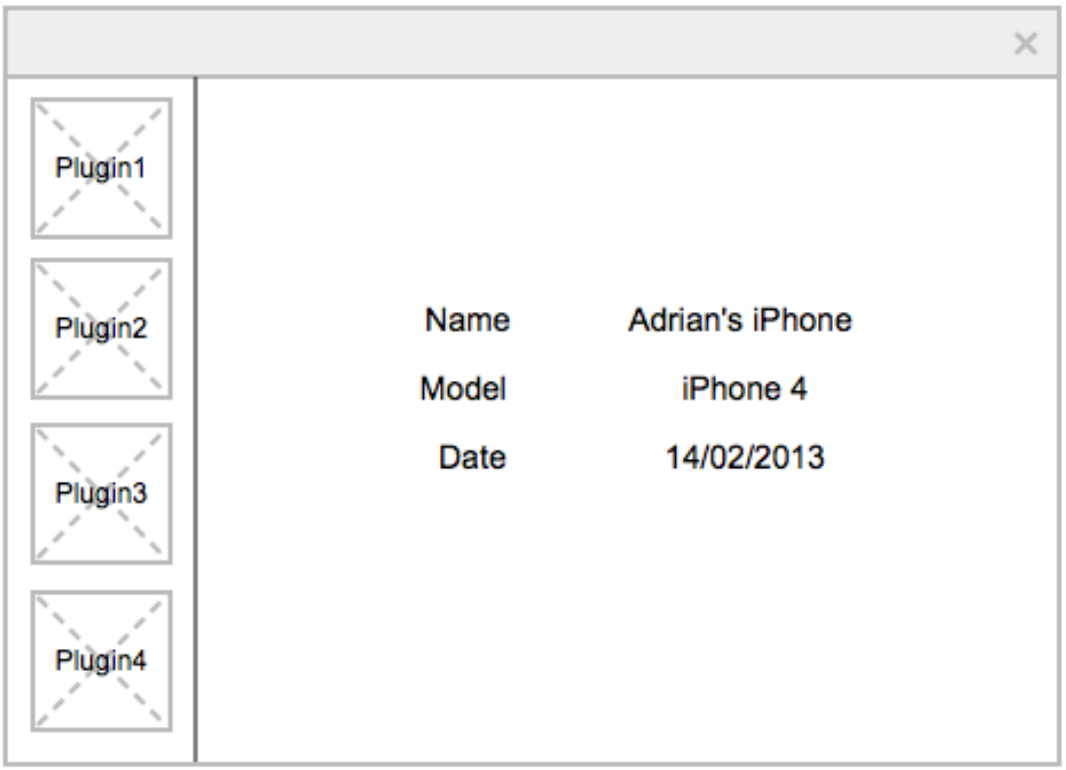

Ilustración 6. Prototipo de plugins

4

Tecnologías utilizadas

La aplicación tiene que ser desarrollada para OS X y para Windows, estas dos son las únicas plataformas para las que se desarrolla iTunes, por lo tanto se busca realizar un único desarrollo que permita su ejecución en ambos sistemas operativos sin tener que hacer dos implementaciones de manera independiente.

4.1 Lenguaje de programación

Se busca un lenguaje de programación que permita un desarrollo rápido y que disponga de un buen conjunto de librerías útiles para tareas como descifrar, ya vengan en las librerías instaladas por defecto o sean desarrolladas por terceros.

Python será el lenguaje de programación elegido. Es interpretado y existen intérpretes para OS X y para Windows. Posee un gran número de librerías de serie y es un lenguaje muy utilizado en el ámbito forense por la cantidad de librerías adicionales de criptografía que existen.

En la actualidad conviven dos versiones de Python claramente diferenciadas, la versión 2.7.5 y la versión 3.3.2. Después de realizar pruebas de lectura de ficheros mbdb se ha concluido que es mucho más fácil la lectura de este tipo de ficheros utilizando Python 2 y requiere un desarrollo extra hacer todo el código compatible con Python 3.

Por las razones anteriores se elige la versión 2 de Python como lenguaje utilizado.

4.2 Interfaz gráfica

La aplicación debe implementarse con una interfaz gráfica multiplataforma. Existen dos opciones: desarrollar una interfaz web o utilizar una librería gráfica.

Desarrollar una interfaz web posibilita hacer una única interfaz para cualquier plataforma en la que se encuentre disponible el intérprete de Python. Además permitiría implementar una arquitectura cliente servidor en la que servidores pudieran realizar toda la carga de descifrado e interpretación de los datos y los clientes con un simple navegador explorar toda la información disponible.

Aunque esta opción parece muy viable, para este proyecto conlleva varios desarrollos adicionales que escapan del objetivo:

- Accesos no autorizados: para que no se pueda acceder a los datos desde cualquier lugar tendría que desarrollarse algún tipo de autenticación que permitiera distinguir los accesos autorizados de los no autorizados. Debido a esta situación tendría que haber un administrador del sistema que se encargue de dar acceso.
- Transferencia de información: tendríamos dos posibilidades, tener la copia de seguridad en el servidor mismo o poder enviarla de manera remota. La primera opción imposibilita el poder hacer un escaneo rápido de la copia de seguridad ya que habría que transportarla físicamente al servidor. La segunda opción puede conllevar un agujero de seguridad importante, los datos viajarían a través de la red por lo que habría que utilizar comunicaciones seguras o cifrar los datos de alguna manera. Además, el rendimiento con esta segunda opción vendrá determinada por el ancho de banda tanto del servidor como del equipo donde se encuentre el cliente.

Debido a que la solución de estos problemas comentados anteriormente escapan del ámbito de este proyecto se optará por utilizar una librería gráfica.

Primeramente quedan descartadas todas aquellas librerías gráficas que no sean compatibles tanto con OS X como con Windows, como, por ejemplo, PyGTK o PyObjc. Utilizar una librería gráfica dependiente de la plataforma provocaría tener que realizar dos desarrollos de manera separada y perder la gran ventaja que proporciona Python.

Tras una extensa búsqueda de interfaces gráficas para Python se proponen tres alternativas, las más utilizadas a día de hoy:

- **Tkinter**: Es la librería que se instala por defecto con el propio intérprete de Python. Fácil de aprender y muy estable.
- **wxPython**: Se trata de un enlace de wxwidgets para Python. Tiene un desarrollo muy activo y una buena comunidad de desarrolladores. Se pueden encontrar muchos ejemplos de código tanto de componentes sencillos como de componentes complejos y muy personalizados.

- **PyQt**: Se trata de un enlace de Qt para Python. También tiene un desarrollo muy activo y se pueden encontrar multitud de ejemplos.

Las características deseables para este proyecto, en orden de importancia, son las siguientes:

- Apariencia nativa: la interfaz ha de adecuarse estéticamente al entorno en el que se está ejecutando.
- Documentación, recurso y ejemplos: buena documentación oficial, gran comunidad de usuarios y recursos y ejemplos disponibles
- Componentes avanzados: ha de tener por defecto controles avanzados como, por ejemplo, selectores de fecha o vistas en árbol.
- Instalación sencilla: la instalación debe ser lo más rápida y sencilla posible para ambas plataformas.

La siguiente tabla muestra una comparativa de características en orden de importancia.

	Tkinter	**wxPython**	**PyQt**
Apariencia nativa	✗	✓	✓
Documentación, recursos y ejemplos	✓	✓	✓
Componentes avanzados	✗	✓	✓
Instalación sencilla	✓	✓	✗

Tabla 5. Comparativa de interfaces gráficas

Después de haber comparado las tres alternativas queda claro que la decisión ha de estar entre wxPython o PyQt porque Tkinter carece de dos de las características mas importantes.

Finalmente se ha elegido wxPython frente a PyQt por la comodidad de la instalación, la documentación y el estilo. Desde la propia página web se puede descargar un binario de wxPython para su instalación en OS X y Windows, mientras que en la página web de PyQt solo se pueden encontrar binarios para Windows. La documentación de PyQt no es tan buena como la de wxPython. La sintaxis de wxPython tiene un estilo más parecido a Python y la de PyQt hereda mucho de C++.

5

Implementación

A continuación se describe cómo ha sido la implementación de la lectura de una copia de seguridad, de las tres pantallas diseñadas y su funcionamiento final. Por último se muestra la implementación de un plugin desarrollado para visualizar el contenido de una base de datos SQLite.

5.1 Localización de copias de seguridad

El resultado final de esta pantalla se muestra en la siguiente captura:

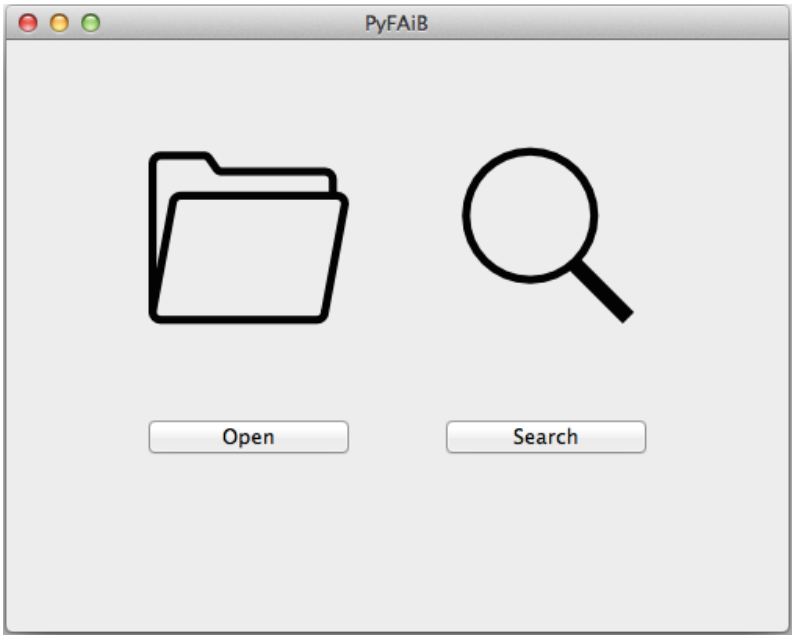

Ilustración 7. Pantalla inicial final

Si pulsamos en el botón “Open” se nos abre un dialogo donde poder navegar por el sistema de ficheros y seleccionar una carpeta. Solo se pasará a la siguiente pantalla si la carpeta seleccionada contiene una copia de seguridad con los archivos necesarios para poder leerla.

Al pulsar en el botón “Search” busca en las carpetas donde, por defecto, iTunes guarda las copias de seguridad. Si encuentra alguna, mostrará un listado donde se podrá elegir la que queramos analizar.

Una vez elegida una copia de seguridad válida se lee y se comprueba si se encuentra cifrada, en caso afirmativo se dirigirá a la pantalla de descifrado, en caso contrario va directo a la pantalla donde se muestran los plugins disponibles.

5.2 Lectura de la copia de seguridad

Para la lectura de una copia de seguridad se leen los ficheros Info.plist, Status.plist, Manifest.plist y Manifest.mbdb.

Los ficheros .plist se encuentran en formato binario y para su lectura se ha utilizado la librería biplist[12]. Provee una función, “readPlist”, a la cual se especifica la ruta de un fichero y devuelve un diccionario con todas las entradas leídas.

Para la lectura del fichero Manifest.mbdb se ha recurrido al código publicado[13] por el usuario galloglass en stackoverflow[14]. A partir de este código se lee cada registro del fichero y se crea un MBDBRecord. Para una mayor legibilidad del código MBDBRecord hereda de la clase dict[15], así para acceder a la ruta original del fichero podemos hacerlo mediante:

```
record[‘path’]
```

[12] https://bitbucket.org/wooster/biplist
[13] http://stackoverflow.com/a/3130860
[14] http://stackoverflow.com/
[15] https://docs.python.org/2/library/stdtypes.html#dict

5.3 Descifrado

La pantalla de descifrado se ha implementado de la siguiente manera:

Ilustración 8. Pantalla de descifrado final

Tenemos tres opciones: introducir la contraseña, realizar un ataque por diccionario o un ataque por fuerza bruta.

En la primera opción solo se prueba con una contraseña por lo que se puede procesar en el mismo hilo que se está ejecutando la interfaz gráfica, pero en las otras dos opciones se ejecutarán procesos que requerirán mucho tiempo de computación, por esta razón es necesario implementar estos ataques utilizando hilos para que la interfaz gráfica pueda seguir respondiendo ante nuestras acciones.

Un ataque por diccionario consiste en intentar averiguar una contraseña probando con las palabras del diccionario. En este caso recurrimos a un listado[16] con las diez mil contraseñas más utilizadas.

[16] https://xato.net/passwords/more-top-worst-passwords/

La arquitectura creada para este tipo de ataque es la siguiente:

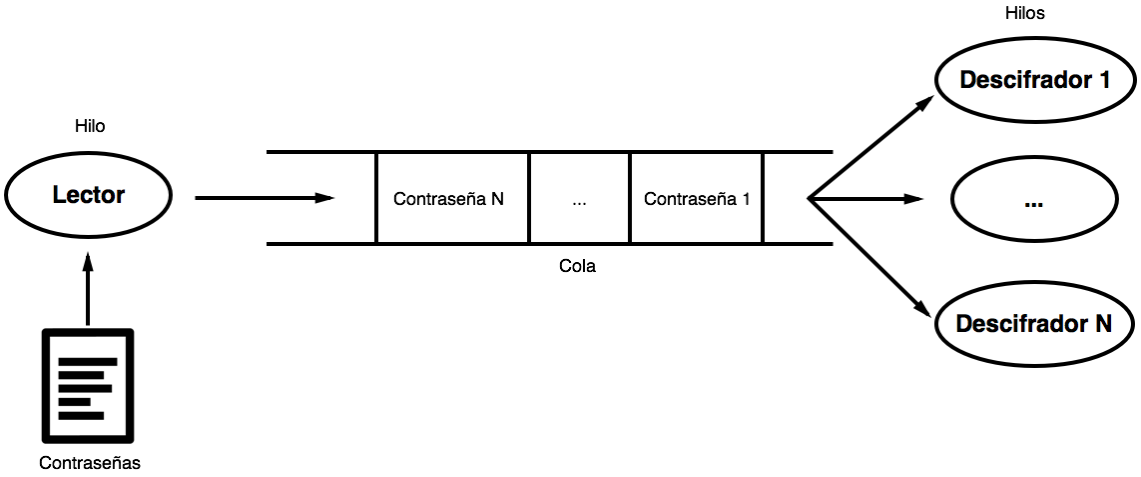

Ilustración 9. Arquitectura del ataque por diccionario

Tenemos un hilo que se encarga de leer una a una las contraseñas del fichero. Cada contraseña que lee la inserta en una cola. Luego se crea una cantidad de hilos que se encargan de extraer contraseñas de la cola e intentar descifrar la copia de seguridad. El número de hilos descifradores se define mediante un parámetro de configuración.

Si los descifradores leyeran directamente del fichero se perderían recursos tratando de sincronizarlos y que no probasen la misma contraseña. Utilizando una estructura de datos como la cola garantizamos que las contraseñas serán probadas en el orden en que vienen escritas en el fichero, esto es muy útil si se ordenan en función de la frecuencia de uso.

Un ataque por fuerza bruta significa intentar averiguar una contraseña probando con todas las posibles combinaciones de letras de cualquier longitud.

La arquitectura creada para el ataque por fuerza bruta es la siguiente:

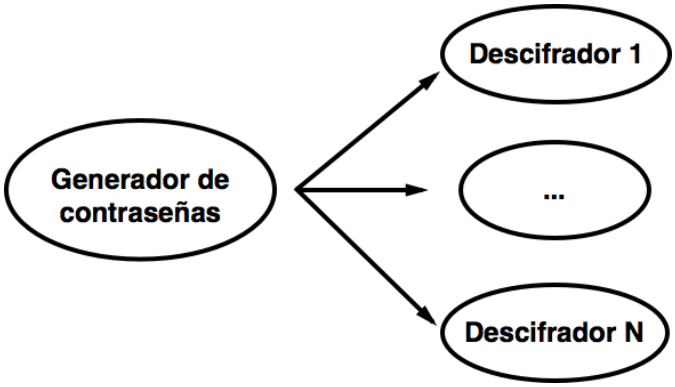

Ilustración 10. Arquitectura del ataque por fuerza bruta

Los descifradores actúan de la misma manera que en el ataque por diccionario, pero en este caso la fuente de las contraseñas es distinta.

Python posee generadores, son funciones que cada vez que son invocadas devuelven un valor que generalmente depende de una llamada anterior. Un pequeño ejemplo en el que se genera la sucesión de Fibonacci ilustrará mejor esta característica:

```
def fibonacci():
    a, b = 0, 1
    while True:
        yield a
        a , b = b, a + b
```

La importancia está en la instrucción “yield”, que en realidad devuelve un objeto al generador y no retorna un valor como podría ser previsible. Pero si iteramos sobre esta función, bien ejecutando su método “next” o ejecutándola en un bucle, conseguimos los valores de la sucesión de Fibonacci: 1, 1, 2, 3, 5, 8, 13, etc.

Visto el anterior ejemplo es más óptimo en cuestión de memoria utilizar un generador para obtener todas las posibles combinaciones de caracteres, éste no almacena todas las palabras que genera en memoria, como podría ser en una lista o un array, sino que las genera a petición y una vez usadas el propio recolector de basura se encarga de liberar memoria como hace con el resto de objetos.

En concreto se ha creado un algoritmo que genera todas las posibles combinaciones de caracteres dados un carácter de inicio y otro de fin. Por defecto itera entre los caracteres ASCII imprimibles, que van del 32 al 126 en números decimales, el carácter 32 es el espacio. A continuación se muestra un ejemplo de la secuencia generada entre los caracteres número 97 y 122, entre la a y la z minúsculas:

```
a, b, c, d … x, y, z, aa, ab, ac, ad … ax, ay, az, ba, bb, bc, bd … aaa, aab, aac,
aad … aax, aay, aaz, aba, abb, abc …
```

Los descifradores llaman al generador cuando necesiten una nueva contraseña y no hay que implementar ningún mecanismo de sincronización para que no prueben la misma contraseña dos veces.

5.4 Vista principal

La vista principal se muestra de la siguiente manera:

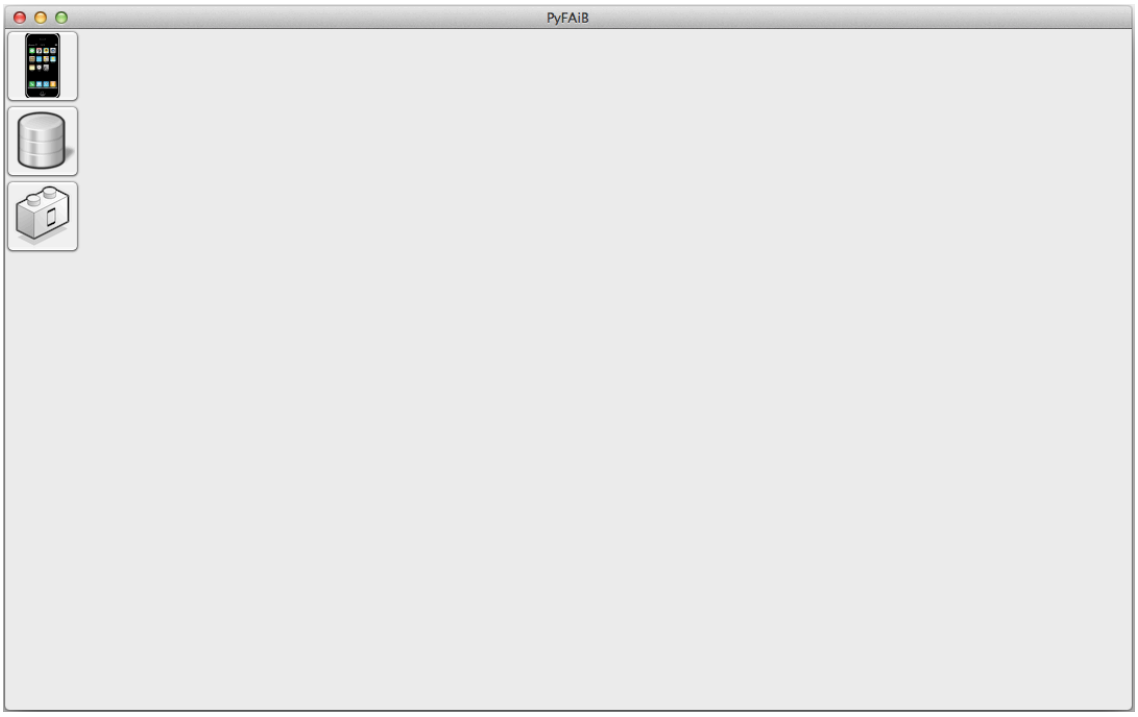

Ilustración 11. Vista principal final

Python posee muchas funciones para realizar introspección. Entre todas las disponibles se han utilizado dos: "__import__" e "issubclass". La primera permite importar módulos dinámicamente y la segunda permite comprobar si una clase hereda de otra.

Con estas utilidades podemos cargar de manera dinámica los plugins existentes. Primero mira en la carpeta de plugins todos los ficheros que acaban con la extensión ".py". Importa cada fichero encontrado, utilizando la instrucción "__import__", y busca si contiene clases que hereden de Plugin con "issubclass". Cada clase que encuentra la añade a un listado que será devuelto finalmente.

Luego, por cada plugin encontrado crea un botón con el icono correspondiente y lo añade al panel lateral, añade el propio plugin a una lista y asocia el evento del botón a mostrar el plugin que le corresponde.

5.5 Plugin

Para mostrar el desarrollo de un plugin se ha creado uno que permite visualizar los contenidos de una base de datos SQLite. Una vez que se elige la base de datos que se

quiere analizar entre todas las que se encuentran en la copia de seguridad se muestra la siguiente pantalla:

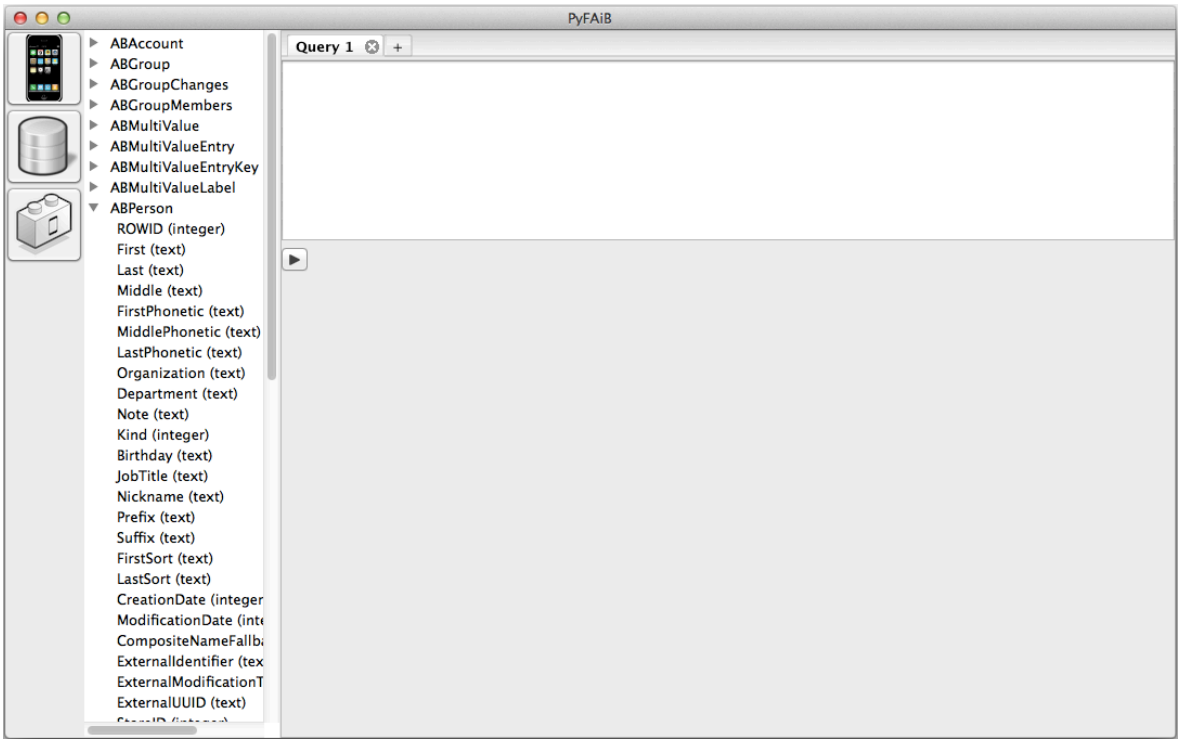

Ilustración 12. Vista principal con plugins final

En el lateral izquierdo se muestra una vista de árbol, donde los elementos de primer nivel son las tablas y sus hijos son las columnas y entre paréntesis el tipo de éstas.

La vista central está compuesta por un controlador con pestañas, cada pestaña tiene una caja de texto en la parte superior para poder escribir consultas que se ejecutarán pulsando el botón inmediatamente inferior y un panel inferior donde se mostrará un error si la consulta no fuera válida o los resultados de ésta en una tabla.

El controlador con pestañas es un elemento personalizado de la interfaz, wxPython trae por defecto el controlador sin un botón para añadir más pestañas, en caso de querer añadirlas tendríamos que crear un botón externo que las crease. Lo que se ha implementado es una pestaña a la que se ha puesto de título "+" para indicar que crea una nueva pestaña y se ha cambiado el comportamiento por defecto al pulsarla, se captura el evento de pulsado sobre ella para ejecutar acciones personalizadas. Al pulsarla se crea una nueva pestaña y se transmite el foco a ella.

A continuación se muestra la pantalla del plugin en la que se ha accedido a la base de datos que contiene los contactos (AddressBook.sqlitedb) y se realiza una consulta para obtener las personas agregadas:

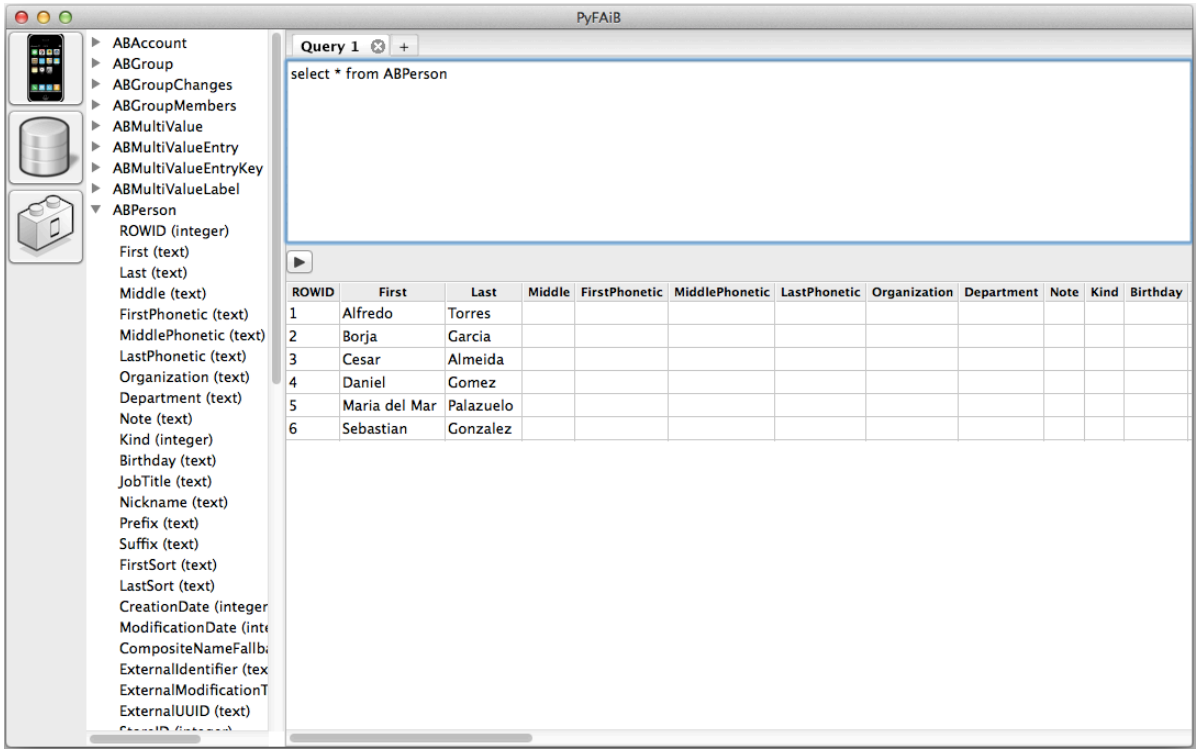

Ilustración 13. Ejemplo de consulta de contactos

6

Conclusión

6.1 Conclusión

En este trabajo se ha desarrollado la arquitectura base de una aplicación con interfaz gráfica que permite localizar, descifrar y analizar copias de seguridad de dispositivos iPhone. Esta arquitectura se ha desarrollado para que sea extensible mediante plugins que permiten al usuario recuperar, analizar o simplemente visualizar los ficheros o la información que necesite y que cualquiera pueda desarrollarlos fácilmente sin tener que preocuparse de tareas como el descifrado de los archivos.

El flujo de trabajo que sigue esta aplicación es primero obtener la localización de una copia de seguridad, luego si se encuentra cifrada el usuario puede introducir la contraseña si la conoce, y si no la conoce tiene la posibilidad de realizar ataques por diccionario o ataques de fuerza bruta para averiguarla y por último se muestran todos los plugins que han sido añadidos para que el usuario pueda utilizarlos.

6.2 Trabajo futuro

Se proponen los siguientes temas como trabajo futuro:

Arquitectura

Se podría desarrollar una arquitectura de comunicación entre plugins que permita intercambiar o, al menos, relacionar información entre plugins.

Un ejemplo sería poder visualizar el historial de mensajes de una aplicación como Whatsapp, pinchar en un número de teléfono y que redirigiera al plugin de contactos para mostrar los datos de dicha persona.

Paralelización

La comprobación de la contraseña para una copia de seguridad cifrada es un proceso muy lento que consume muchos recursos. La aplicación es capaz de crear un número determinado de hilos, especificado mediante configuración, que prueban una serie de contraseña, pero la ejecución de estos hilos viene determinada por el GIL, que no aprovecha bien todos los recursos de la CPU. Sería interesante desarrollar un mecanismo por el cual se pudiera comunicar con servidores de alto rendimiento que ejecutaran Hadoop, esto haría mucho más rápido el proceso de averiguar la contraseña.

Plugins

Ya que la arquitectura nos permite añadir funcionalidad de manera rápida y sencilla se puede seguir desarrollando diferentes plugins que extraigan más información de la copia de seguridad.

Se pueden identificar dos tipos de plugins a desarrollar: aquellos que extraen información que almacena el propio dispositivo y los que extraen información de aplicaciones específicas.

Ejemplos:

- Reconocimiento facial en fotografías que permita identificar personas con las que haya podido estar.
- Recolección de datos Exif de fotografías para generar un histórico de localización.
- Descarga y búsqueda de información de las redes sociales utilizadas sin tener que acudir al navegador y buscar manualmente.

Interfaz gráfica

Se ha optado por la utilización de una interfaz gráfica de escritorio pero de igual manera se puede desarrollar una interfaz web y así disponer de ambas alternativas. De esta manera la comunicación entre plugins sería muy fácil, cada uno podría hacer públicas una serie de urls con los parámetros que aceptase.

Búsqueda de copias de seguridad

Se podría automatizar la búsqueda de copias de seguridad en todo un dispositivo de almacenamiento y poder realizar ataques en aquellas que estén cifradas para descubrir su contraseña.

Internacionalización

Todos los textos de la interfaz han sido escritos en inglés, pero sería bueno exportar todos los textos a plantillas de texto traducibles para que sea mas sencilla su traducción.

Glosario

· **AES:** Advanced Encryption Standard

· **ASCII:** American Standard Code for Information Inerchange

· **CBC:** Cipher-Block Chaining

· **Exif:** Exchangeable image file format

· **IMEI:** International Mobile Equipment Identity

· **PBKDF2:** Password-Based Key Derivation Function 2

· **SHA:** Secure Hash Algorithm

· **UDID:** Unique Device Identifier

· **UUID:** Universally Unique Identifier

Referencias

iTunes Backup – The iPhone Wiki
http://theiphonewiki.com/wiki/ITunes_Backup

iPhone Forensics – Analysis of iOS 5 Backups
http://www.securitylearn.net/2012/05/06/iphone-forensics-analysis-of-ios-5-backups/iphone-dataprotection
http://www.securitylearn.net/tag/understand-ios-backups-decrypt-iphone-backup-with-known-password-itunes-backup-in-detail/

iPhone backup – mbdb file structure
http://www.securitylearn.net/tag/ios-backup-mbdb-file-format/iphone-dataprotection

Processing iPhone Backup Files
http://www.appleexaminer.com/iPhoneiPad/iPhoneBackup/iPhoneBackup.html

iPhone data protection tools
https://code.google.com/p/iphone-dataprotection/

Python GUI Programming
https://wiki.python.org/moin/GuiProgramming

Graphic User Interface FAQ
https://docs.python.org/2/faq/gui.html

Choosing wxPython over Tkinter
http://wiki.wxpython.org/Choosing%20wxPython%20over%20Tkinter